AMH: a new Framework to Design Adaptive Metaheuristics

Aymeric Blot, Marie-Éléonore Kessaci-Marmion, Laetitia Jourdan

AMH: a new Framework to Design Adaptive Metaheuristics

Aymeric Blot, Marie-Éléonore Kessaci-Marmion, Laetitia Jourdan

Université de Lille, Inria, CNRS, UMR 9189 – CRIStAL, France
aymeric.blot@inria.fr
{laetitia.jourdan,me.kessaci}@univ-lille1.fr

Abstract

Metaheuristics should be configured to perform well on a given problem. Their configuration is either made off-line by automatic algorithm configuration tools or on-line with control mechanisms to adapt their behaviour. The former requires a flexible structure that may be modified during the execution. Therefore, the implementation of such a structure is not straightforward to enable modifications of optimisation strategies and not of parameter values only. In this work, we present AMH, a framework dedicated to the design of configurable metaheuristics. This framework is based on controlling the execution flow of metaheuristics to enable the implementation of flexible structures.

1 Introduction

Metaheuristics often expose multiple parameters that strongly impact their performance. Indeed, to achieve good performance, the value of their parameters needs to be specifically set. Off-line configuration can be efficiently done with automatic algorithm configuration tools. On the other hand, on-line configuration is generally done through adaptive metaheuristics where intern control mechanisms modify the algorithms during the execution.

The implementation of such adaptive algorithms is not straightforward and become more and more difficult when the control is applied on optimisation strategies. Therefore, we propose a framework, called AMH, dedicated to the design and configuration of metaheuristics.

First, Section 2 explains the motivations that have led us to propose a new framework. Then, Section 3 presents our framework. Finally, Section 4 concludes this paper and draws perspectives.

2 Motivation to Propose a New Framework

The performance of a metaheuristic depends on its parameters being numerical values (*e.g.*, thresholds, counters, probabilities) or categorical values. The latter represent components such as the different recombination operators of EAs or the neighbourhood operator in SLS, or the different strategies of a metaheuristic such as the restart or the kick-move in the perturbation of an iterated local search.

Metaheuristics have recently benefited from automatic algorithm configuration with tools such as irace [1], ParamILS [2], or SMAC [3]. It corresponds to off-line configuration where the parameters are settled before the final execution. When parameters are numerical values, these tools may be directly and easily plugged with the executable program, the single requirement being to be configurable from the command line. When parameters are optimization strategies, it is not straightforward, but possible using automatic code generation to test different configurations. However, each generated executable program embeds one configuration of the metaheuristic only. We call these algorithms *static* metaheuristics, opposite to *adaptive* metaheuristics. They integrate control mechanisms enabled to modify the values of the parameters set initially as well as the optimisation strategies in order to adapt more deeply to the problem instance being solved [1, 3]. On-line configuration is based on these control mechanisms that apply minor modifications (parameter values) or major modifications (use of strategy). Therefore, an

[1] `http://iridia.ulb.ac.be/irace/`
[2] `www.cs.ubc.ca/labs/beta/Projects/ParamILS/`
[3] `http://www.cs.ubc.ca/labs/beta/Projects/SMAC/`

adaptive metaheuristic requires a flexible structure, as every parameter, every strategy or the algorithm itself may be modified/adapted during the execution. Moreover, the control mechanisms are based on feedback measures to evaluate the relevance of a used mechanism and an update mechanism to setup the algorithm for the next iteration. The implementation of such a structure with the necessary feedback and update tools is not straightforward.

ParadisEO [4] (C++) and jMetal [5] (java) are well-known frameworks of the literature dedicated to the design of metaheuristics, which can be used with automatic algorithm configuration tools [4] to obtain optimised algorithms for given problems. While it is possible for both of them to handle on-line configuration when only the value of numerical parameters are adapted during the execution of the algorithm, their design do not take into account the possibility to fully modify an algorithm during its execution. With the same intention to propose a useful framework to design metaheuristics, we propose a new framework that enable both on-line and off-line configuration of metaheuristics.

3 AMH Framework

AMH is a framework designed to implement both static and adaptive metaheuristics. The main goal of AMH is to enable the on-line design of algorithm whose structure can be modified during the execution.

3.1 Philosophy of AMH

A metaheuristic can be seen as a succession of individual statements, instructions or function calls. Hence, a metaheuristic can be associated to a specific flow of execution. For example, the execution flow of a GA is composed of 3 successive strategy components (selection, crossover and then mutation) followed with a termination component that loop the process until the termination criterion is met. Every component of the algorithm can be seen as a function taking as input and output a population of solution. Obviously, considering any other metaheuristic, the input might be a unique solution or a Pareto archive in multi-objective optimisation. Any coherent part of the GA (and thus, the whole algorithm itself) can be seen as such a function: the crossover and the mutation could be aggregated in a single component returning new solutions, that could itself be aggregated with the selection component. In fact, the entire contents of the loop can be considered as a simple function updating the population of solution.

Setting parameter values of a metaheuristic does not interfere with the execution. On the other hand, the setting of the optimisation strategies determines the execution flow. For a static metaheurictics, this setting is done before the run and so does not modify the execution flow during the execution of the algorithm whereas the execution flow of an adaptive metaheurictics may be fully modified. For example, considering an adaptive GA, we can imagine that a control mechanism chooses to delete the mutation component, or replace it with a local search component, from a certain moment of the execution that is to say that the execution flow is clearly modified during the execution.

AMH has been designed to control the execution flow before and during the execution of the metaheuristic. The graph of the execution flow is created at the start of the algorithm, hence facilitating algorithm configuration for parameters modifying its structure. This graph being managed and traversed by AMH and not compiled, it becomes possible to modify it during its own execution, allowing a natural control over its components, and more generally speaking, allowing on-line configuration.

3.2 Design and Implementation of AMH

The AMH framework[6] is implemented in C++. It handles the execution flow of a given algorithm by encapsulating algorithmic operations in a meta-component that describes the temporal interactions between components. All algorithms implemented in AMH inherit from a base function class – a single

[4] http://paradiseo.gforge.inria.fr/
[5] https://jmetal.github.io/jMetal/
[6] http://github.com/amh-framework

class representing a function – *i.e.*, a delimited part of the execution having defined input and output types, which are specified at compile-time using templates. Moreover, AMH provides a large range of *execution flow primitives* such as conditions and loops, in order to connect all parts of an implemented algorithm.

The core design of AMH is to focus only on the flow of execution, and not on the solving methods. Indeed, the algorithm designer has to provide the solution representations, solving mechanisms, and algorithm structure, or to use existing ones. The solution representations are used in template at compile-time, and solving mechanisms need to be encapsulated, either as static classes inheriting from the base AMH function class, or dynamically as native C++ functions. In particular, it means that existing C++ algorithm implementations (*e.g.*, metaheurictics implemented under ParadisEO) can benefit from AMH just by defining atomic components (*e.g.*, selection and variation strategies of a GA) and encapsulating them. Hybridisation of algorithms using the same solution representation is immediate, and the execution flow of AMH provides easy algorithm designs enabled off-line and on-line configuration.

The base class of AMH is `amh::algo<IN,OUT>` representing a function taking an argument of class `IN` and outputing an argument of class `OUT`, generally identical and encoding the current state of the metaheuristic. All AMH classes use similar templates, enforcing the validity of the resulting algorithm. AMH also provides multiple useful subclasses, such as `amh::gen` to implement generators such as random solution generators or neighbourhoods, `amh::func` to encapsulate native C++ functions directly without having to create ad hoc classes, or likewise `amh::check` for Boolean formula. Moreover, AMH provides useful classes, as for example `if-then-else`, `while`, `do-while`, `until`...

AMH has been successfully used to off-line configure multi-objective local search (MOLS) algorithms [2] where 2790 configurations were considered. Control mechanisms such as multi-armed bandit or adaptive pursuit have been integrated to AMH. Experiments have also shown the interest of using AMH to on-line configure efficient MOLS.

4 Conclusion

Both off-line and on-line configuration is used to adapt an algorithm to a given problem. While off-line configuration can be done by plug-and-play with automatic algorithm configuration tools, on-line configuration requires algorithm-specific components and a flexible structure. In this paper, we have presented our framework AMH designed and implemented to facilitate on-line configuration of metaheuristics, *i.e.*, the design of adaptive metaheuristics. Hence, we showed that it is easy to integrate control mechanisms into metaheuristics that modify parameter values or algorithm strategies. These are made possible with AMH as it dynamically creates and handles the execution flow of the algorithm whose instantiation can be adapted during its execution.

Following this work, we aim at instantiating many strategies of metaheuristics from bio-inspired algorithms as well as neighbourhood-based algorithms into AMH to design new adaptive metaheuristics. Works on adaptive metaheuristics remain few in number but AMH gives some implementation hints that will be investigated to design control mechanisms for the whole execution flow.

References

[1] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations Research/Computer Science Interfaces Series*. Springer, 2009.

[2] A. Blot, A. Pernet, L. Jourdan, M-E. Kessaci-Marmion, and H. H. Hoos. Automatically configuring multi-objective local search using multi-objective optimisation. In *EMO 2017*, 2017.

[3] L. Da Costa, Á. Fialho, M. Schoenauer, and M. Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *GECCO 2008*, pages 913–920. ACM Press, 2008.

[4] M-E. Marmion, F. Mascia, M. López-Ibáñez, and T. Stützle. Automatic design of hybrid stochastic local search algorithms. In *HM 2013*, pages 144–158, LNCS, 2013.